Licence d'informatique
L313

Jeu d'instructions NIOS II

1 Formats d'instructions

Le processeur NIOS II a un jeu d'instructions de type RISC. Il possède 32 registres de 32 bits, notés r0 à r31, avec r0 ≡ 0.
Les instructions sont de longueur fixe (32 bits). Il y a trois formats d'instructions.

1. Le format I est indiqué en Figure 1. Le code opération est sur 6 bits. Les champs A et B sur 5 bits contiennent un numéro de registre. Dans la plupart des cas, A indique le numéro du registre source et B le numéro du registre destination. IMM16 est un immédiat sur 16 bits.

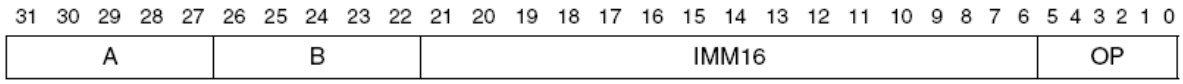

Figure 1 : Instructions de type I

2. Le format R est donné en Figure 2. Le code opération est sur 6 bits. Une extension du code opération est codée sur 11 bits. A et B contiennent les numéros des registres source et C contient le numéro du registre destination.

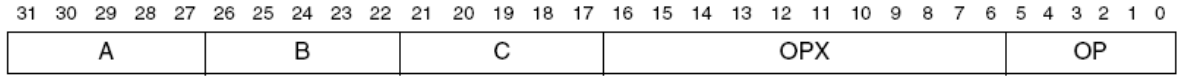

Figure 2 : Instructions de type R

3. Le format J est donné en Figure 3. Le code opération est sur 6 bits et IMMED26 est un immédiat sur 26 bits. Seule l'instruction CALL utilise ce format.

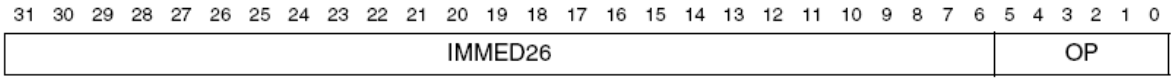

Figure 3 : Instructions de type J

2 Les différents types d'instructions

2.1 Les instructions mémoire

Les instructions load (chargement) et store (rangement) transfèrent des données entre la mémoire (ou les entrées/sorties) et les registres du processeur. Elles utilisent le format I.
La mémoire est adressable par octet. Les adresses mémoire sont calculées comme la somme du contenu d'un registre (rA) et de simm16, qui est l'extension de signe sur 32 bits de IMM16.
Les instructions sont données dans la table. Les instructions load et store ont des versions spéciales pour les composants d'entrée/sortie. La différence entre les versions mémoire et les versions entrées/sorties, repérées par l'ajout de io, est que ces dernières contournent le cache de données lorsque celui-ci est présent.

ldw / ldwio	ldw rB, imm16(rA)	rB ← Mem32 (rA+simm16)
ldb /ldbio	ldb rB, imm16(rA)	rB ← ES | Mem8 (rA+simm16)
ldbu /ldbuio	ldbu rB, imm16(rA)	rB ← Zero | Mem8 (rA+simm16)
ldh /ldhio	ldh rB, imm16(rA)	rB ← ES | Mem16 (rA+simm16)

ldbu /ldbuio	ldhu rB, imm16(rA)	rB ← Zero | Mem16 (rA+simm16)
stw /stwio	stw rB, imm16(rA)	Mem32 (rA+simm16) ← rB
stb /stbio	stb rB, imm16(rA)	Mem8 (rA+simm16) ← 8 bits de poids faible de rB
sth /sthio	sth rB, imm16(rA)	Mem16 (rA+simm16) ← 16 bits de poids faible de rB

2.2 Les instructions arithmétiques

Les instructions arithmétiques opèrent sur des opérandes contenus dans des registres (format R) ou un opérande situé dans un registre et un immédiat (format I). Les immédiats sont étendus sur 32 bits (extension de signe).
A noter que selon les versions du processeur NIOS II, les instructions de multiplication et de division sont implantées soit par logiciel, soit par matériel.

add	add rC,rA,rB	rC ← rA + rB
addi	addi rB,rA,imm16	rB ← rA + simm16
sub	sub rC,rA,rB	rC ← rA + rB
subi	subi rB,rA,imm16	rB ← rA - simm16 (pseudo : addi rB, rA, -imm16)
mul	mul rC,rA,rB	rC ← rA * rB (32 bits poids faible du résultat)
muli	muli rB,rA,imm16	rB ← rA * simm16 (32 bits poids faible du résultat)
div	div rC,rA,rB	rC ← rA / rB - rA et rB contiennent des nombres signés
divu	divu rC,rA,rB	rC ← rA / rB - rA et rB contiennent des nombres non signés

2.3 Les instructions logiques

Les instructions logiques opèrent sur des opérandes contenus dans des registres (format R) ou un opérande situé dans un registre et un immédiat (format I). Les immédiats sont étendus sur 32 bits (extension de zéros, notée zimm16).
Les instructions andhi, orhi, xorhi utilisent un opérande immédiat sur 32 bits en utilisant imm16 comme bits de poids forts et 16 zéros comme bits de poids faible.

and	and rC,rA,rB	rC ← rA et rB (ET logique bit à bit)
andi	andi rB,rA,imm16	rB ← rA et zimm16
or	or rC,rA,rB	rC ← rA ou rB (OU logique bit à bit)
ori	ori rB,rA,imm16	rB ← rA ou zimm16
xor	xor rC,rA,rB	rC ← rA xor rB (OU exclusif bit à bit)
xori	xori rB,rA,imm16	rB ← rA xor zimm16
nor	nor rC,rA,rB	rC ← rA nor rB (NOR bit à bit)
nori	nori rC,rA,rB	rB ← rA nor zimm16
andhi	andhi rB,rA,imm16	rB ← rA et imm16|0000000000000000
orhi	orhi rB,rA,imm16	rB ← rA or imm16|0000000000000000
xorhi	xorhi rB,rA,imm16	rB ← rA xor imm16|0000000000000000

2.4 Les instructions de transfert

Les instructions de transfert transfèrent le contenu d'un registre ou un immédiat étendu dans un autre registre. Elles sont implantées à l'aide de pseudo-instructions

mov	mov rC,rA	rC ← rA réalisé par add rC,rA,r0
movi	movi rB,imm16	rB ← simm16 réalisé par addi rB, r0, imm16
movui	movui rB,imm16	rB ← zimm16 réalisé par ori rB, r0, imm16

Le transfert d'une adresse de 32 bits dans un registre est implanté par la pseudo-instruction movia rB, ETIQ où ETIQ est une adresse sur 32 bits. Elle est implantée par les deux instructions

```
orhi rB, ro, %hi (ETIQ)      // %hi(ETIQ) correspond aux 16 bits de poids fort
or rB,r0, %lo(ETIQ)          //%lo(ETIQ) correspond aux 16 bits de poids faible
```

2.5 Instructions de comparaison

Les instructions de comparaison comparent le contenu de deux registres (format R) ou le contenu d'un registre et d'un immédiat (étendu sur 32 bits) et écrit un 1 (vrai) ou un 0 (faux) dans le registre résultat. Dans le tableau ci-dessous, les instructions en italique sont des pseudo-instructions, implantées par l'assembleur à l'aide d'autres instructions de comparaison et en permutant le contenu des registres pour les instructions de type R, et en modifiant l'immédiat pour les instructions de type I. Dans la table ci-dessous, les instructions en italique sont des pseudo-instructions.

cmplt	cmplt rC,rA, rB	si rA < rB (signés) rC ← 1 sinon rC ← 0
cmpltu	cmpltu rC,rA, rB	si rA < rB (non signés) rC ← 1 sinon rC ← 0
cmpeq	cmpeq rC,rA, rB	si rA == rB (signés) rC ← 1 sinon rC ← 0
cmpne	cmpne rC,rA, rB	si rA != rB (signés) rC ← 1 sinon rC ← 0
cmpge	cmpge rC,rA, rB	si rA >= rB (signés) rC ← 1 sinon rC ← 0
cmpgeu	cmpgeu rC,rA, rB	si rA >= rB (non signés) rC ← 1 sinon rC ← 0
cmpgt	*cmpgt rC,rA, rB*	*si rA > rB (signés) rC ← 1 sinon* rC ← *0*
cmpgtu	*cmpgtu rC,rA, rB*	*si rA > rB (non signés) rC ← 1 sinon* rC ← *0*
cmple	*cmple rC,rA, rB*	*si rA <= rB (signés) rC ← 1 sinon* rC ← *0*
cmpleu	*cmpleu rC,rA, rB*	*si rA <= rB (non signés) rC ← 1 sinon* rC ← *0*
cmplti	cmplti rC,rA,IMM16	si rA <simm16 (signés) rC ← 1 sinon rC ← 0
cmpltui	cmpltui rC,rA,IMM16	si rA < zimm16 (non signés) rC ← 1 sinon rC ← 0
cmpeqi	cmpeqi rC,rA,IMM16	si rA == simm16 (signés) rC ← 1 sinon rC ← 0
cmpnei	cmpnei rC,rA,IMM16	si rA != simm16 (signés) rC ← 1 sinon rC ← 0
cmpgei	cmpgei rC,rA,IMM16	si rA >= simm16 (signés) rC ← 1 sinon rC ← 0
cmpgeui	cmpgeui rC,rA,IMM16	si rA >= zimm16 (non signés) rC ← 1 sinon rC ← 0
cmpgti	*cmpgti rC,rA,IMM16*	*si rA > simm16 (signés) rC ← 1 sinon* rC ← *0*
cmpgtui	*cmpgtiu rC,rA,IMM16*	*si rA > zimm16 (non signés) rC ← 1 sinon* rC ← *0*
cmplei	*cmplei rC,rA,IMM16*	*si rA <= simm16 (signés) rC ← 1 sinon* rC ← *0*
cmpleui	*cmpleui rC,rA,IMM16*	*si rA <= zimm16 (non signés) rC ← 1 sinon* rC ← *0*

2.6 Instructions de décalage

Les instructions de décalage décalent le contenu d'un registre vers la droite ou vers la gauche et placent le résultat dans un autre registre. Dans le cas du décalage à droite, le décalage logique introduit des zéros à gauche alors que le décalage arithmétique réintroduit le bit de signe. Le décalage à gauche introduit des zéros à droite.
Toutes les instructions de décalage sont de type R.

Le nombre de décalages est spécifié soit par les 5 bits de poids faible du registre rB, soit par un immédiat sur 5 bits contenu dans l'extension du code opération du format R.

srl	srl rC,rA, rB	rC ← rA >> (nb spécifié dans rB) - Logique
srli	srli rC,rA, imm5	rC ← rA >> (imm5) - Logique
sra	srl rC,rA, rB	rC ← rA >> (nb spécifié dans rB) - Arithmétique
srai	srai rC,rA, imm5	rC ← rA >> (imm5) - Arithmétique
sll	srl rC,rA, rB	rC ← rA << (nb spécifié dans rB)
slli	slli rC,rA, imm5	rC ← rA << (imm5) - Logique

3 Instructions de rotation

Il y a trois instructions de rotation, qui utilisent le format R. Pour ces instructions, une rotation à droite décale à droite les bits d'un registre en réintroduisant à gauche les bits sortant à droite et place le résultat dans un autre registre. Le nombre de rotations est spécifié comme le nombre de décalages par les cinq bits de poids faible de rB ou par un immédiat de 5 bits contenu dans l'extension du code opération de l'instruction.

ror	ror rC,rA, rB	rC ← rotation droite de rA (nb spécifié dans rB)
rol	rol rC,rA, rB	rC ← rotation gauche de rA (nb spécifié dans rB)
roli	rol rC,rA, imm5	rC ← rotation gauche de rA (imm5)

4 Instructions de saut et de branchement.

L'instruction de saut jmp rA effectue un saut à l'adresse dans rA.
L'instruction de branchement inconditionnel br ETIQ branche à l'adresse ETIQ. C'est une instruction de type I. La valeur imm16 correspond au déplacement signé entre l'adresse de l'instruction qui suit l'instruction br et l'adresse cible du branchement (ETIQ).
Les instructions de branchement conditionnel comparent le contenu de deux registres et effectuent le branchement si la condition est vraie ou exécutent l'instruction suivante si la condition est fausse.
Dans la table ci-dessous

- NPC est l'adresse de l'instruction suivant le branchement, soit Adresse de l'instruction de branchement + 4
- simm16 = Adresse cible (ETIQ) – NPC
- Les instructions en italique sont des pseudo-instructions

blt	blt rA,rB, ETIQ	PC ← NPC + simm16 si rA<rB (signés) ; PC ← NPC sinon
bltu	bltu rA,rB, ETIQ	PC ← NPC + simm16 si rA<rB (non signés) ; PC ← NPC sinon
beq	beq rA,rB, ETIQ	PC ← NPC + simm16 si rA==rB (signés) ; PC ← NPC sinon
bne	bne rA,rB, ETIQ	PC ← NPC + simm16 si rA!=rB (signés) ; PC ← NPC sinon
bge	bge rA,rB, ETIQ	PC ← NPC + simm16 si rA>=rB (signés) ; PC ← NPC sinon
bgeu	bgeu rA,rB, ETIQ	PC ← NPC + simm16 si rA>=rB (non signés) ; PC ← NPC sinon
bgt	*bgt rA,rB, ETIQ*	*PC ← NPC + simm16 si rA>rB (signés) ; PC ← NPC sinon*
bgtu	*bgtu rA,rB, ETIQ*	*PC ← NPC + simm16 si rA>rB (non signés) ; PC ← NPC sinon*
ble	*ble rA,rB, ETIQ*	*PC ← NPC + simm16 si rA<=rB (signés) ; PC ← NPC sinon*
bleu	*bleu rA,rB, ETIQ*	*PC ← NPC + simm16 si rA<=rB (non signés) ; PC ← NPC sinon*

5 Instructions d'appel et retour de fonctions

Il y a deux instructions d'appel de fonctions (sous programmes).
L'instruction call ETIQ est de type J. Elle contient un immédiat sur 26 bits. L'instruction call effectue les actions suivantes :

- Elle range l'adresse de retour (NPC) dans le registre r31.
- Elle saute à l'adresse ETIQ, qui est obtenue par concaténation des 4 bits de poids fort de l'adresse de CALL, des 26 bits de l'immédiat et de 00 (Les deux zéros garantissent que les adresses sont alignées sur des frontières de mots de 32 bits).

L'instruction callr rA effectue les actions suivantes

- Elle range l'adresse de retour (NPC) dans le registre r31.
- Elle saute à l'adresse contenue dans rA.

Le retour de fonction (sous programme) est réalisé par l'instruction ret. C'est une pseudo-instruction qui correspond à jmp r31.

6 Utilisation des registres

Le processeur NIOS II a 32 registres de 32 bits. Certains registres ont un rôle particulier, et ont un nom spécial reconnu par l'assembleur.

- Le registre r0 contient la constante 0. On ne peut écrire dans ce registre. Il est aussi appelé `zero`
- Le registre r1 est utilisé par l'assembleur comme registre temporaire. Il ne doit pas être utilisé dans les programmes utilisateur.
- Les registres r24 et r29 sont utilisés pour le traitement des exceptions. Ils ne sont pas disponibles en mode utilisateur
- Les registres r25 et r30 sont utilisés exclusivement par le module de débogage JTAG.
- Les registres r27 et r28 sont utilisés pour contrôler la pile
- Le registre r31 est utilisé pour les adresses de retour des fonctions/sous programmes.

Registre	Nom	Fonction
R0	Zero	0x00000000
R1	at	Temporaire pour l'assembleur
R2		
.		
R23		
R24	et	Temporaire pour le traitement des exceptions
R25	bt	Temporaire pour les points d'arrêt
R26	gp	Pointeur global
R27	sp	Pointeur de pile (stack pointer)
R28	fp	Pointeur de trame (Frame pointer)
R29	ea	Adresse de retour des exceptions
R30	ba	Adresse de retour des points d'arrêt
R31	ra	Adresse de retour

Figure 4 : utilisation et nom des registres